A First-Order Semantics for Golog and ConGolog under a Second-Order Induction Axiom for Situations

Fangzhen Lin
Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong

Abstract

Golog and ConGolog are languages defined in the situation calculus for cognitive robotics. Given a Golog program δ, its semantics is defined by a macro $Do(\delta, s, s')$ that expands to a logical sentence that captures the conditions under which performing δ in s can terminate in s'. A similar macro is defined for ConGolog programs. In general, the logical sentences that these macros expand to are second-order, and in the case of ConGolog, may involve quantification over programs. In this paper, we show that by making use of the foundational axioms in the situation calculus, in particular, the second-order closure axiom about the space of situations, these macro expressions can actually be defined using first-order sentences.

Introduction

Golog [Levesque et al., 1997] and ConGolog [De Giacomo, Lespérance, and Levesque, 2000] are Algol-like programming languages defined in the situation calculus [McCarthy, 1968; Reiter, 2001]. They are intended for high-level control of agents, including both physical robots and virtual agents [Reiter, 2001; Burgard et al., 1999; Funge, 2000; Lespérance, Levesque, and Ruman, 1997; McIlraith and Son, 2002]. They include non-determinism, and in the case of ConGolog, concurrency.

The semantics of Golog programs is axiomatized in logic by macro expansions [Levesque et al., 1997]: given a Golog program δ and two situation terms s and s', the macro expression $Do(\delta, s, s')$ is used to denote that s' is a terminating situation of performing δ in s. There may be more than one possible terminating situation as the program may be indeterminate. In general, the macro expression $Do(\delta, s, s')$ expands into a second-order sentence because δ may have loops. In the case of ConGolog programs, which are extensions of Golog programs with some concurrency operators, De Giacomo, Lespérance and Levesque [2000] introduced a one-step transition predicate $Trans(\delta, s, \delta', s')$ (performing δ in s for one step may end up in situation s' with δ' as the remaining program to be performed) and a final predicate $Final(\delta, s)$ (program δ terminates in s). The Do macros are then defined using these predicates, and in general expand to sentences that are not only second-order, but also involve quantification over programs.

The second-order formulas that are needed in Golog and ConGolog macro expansions are fundamentally the same as the ones for defining transitive closures. One of the foundational axioms of the situation calculus is a second-order one that defines the space of situations as the transitive closure of $do(a, s)$ starting at the initial situation S_0. In this paper, we show that this second-order foundational axiom can be used to capture loops and recursions so that for Golog and ConGolog programs, the macro expression $Do(\delta, s, s')$ can in fact be defined using first-order sentences.

The situation calculus

We first briefly review the situation calculus used for Golog and ConGolog. For more details, see [Reiter, 2001; Lin, 2007]. The language of the situation calculus is a many-sorted one with sorts $action$ for actions, $situation$ for situations, and other domain dependent ones. There is a constant S_0 of sort $situation$, and a function $do : action \times situation \rightarrow situation$.

The foundational axioms say that the set of situations consists exactly of those that can be constructed from S_0 using function $do(a, s)$ (we assume that variables in displayed formulas are universally quantified from outside):

$$S_0 \neq do(a, s), \tag{1}$$

$$do(a_1, s_1) = do(a_2, s_2) \supset a_1 = a_2 \wedge s_1 = s_2, \tag{2}$$

$$\forall P.[P(S_0) \wedge \forall a, s(P(s) \supset P(do(a, s)))] \supset \forall s P(s), \tag{3}$$

The first two axioms are unique names axioms, and the third one a second-order induction axiom. Given a sequence $[a_1, ..., a_n]$ of actions, we use $do([a_1, ..., a_n], s)$ to denote the situation resulting from doing the sequence of actions in s: $do(a_n, ..., do(a_1, s)...)$. Using this notation, we can understand the foundational axioms as saying that s is a situation iff there is a sequence α of actions such that $s = do(\alpha, S_0)$.

With these axioms, one can define other relations inductively on situations [Reiter, 1993; Lin and Reiter, 1994]. The following axioms define a partial order $\leq$ on situations:

$$\neg s < S_0,$$
$$s < do(a, s') \equiv s = s' \vee s < s',$$
$$s \leq s' \equiv s < s' \vee s = s'.$$

Again in terms of sequences of actions, we can see that $s \leq s'$ iff there is a sequence α of actions such that $s' = do(\alpha, s)$.

In the following, we denote by Σ the set of foundational axioms as well as the above axioms for $<$.

Golog

In this section, we briefly review the definitions of Golog programs and their semantics as defined in [Levesque et al., 1997]. Golog programs are complex actions and procedures. Complex actions are defined inductively as follows:

$$\begin{array}{rcl}\delta & ::= & A \mid \varphi? \mid \delta;\delta \mid \\ & & (\delta \mid \delta) \mid (\pi x)\delta \mid \delta^*\end{array}$$

where A is a primitive action, φ is a pseudo-situation calculus formula with all situation arguments suppressed. In [Levesque et al., 1997], the semantics of a complex action δ is defined by a macro $Do(\delta, s, s')$ that expands to a situation calculus formula that says that s' is a terminating situation of performing δ in s. Formally, this macro expansion is defined inductively as follows:

$$\begin{array}{l}Do(A, s, s') \stackrel{def}{=} Poss(A, s) \wedge s' = do(A, s),\\ Do(\varphi?, s, s') \stackrel{def}{=} s' = s \wedge \varphi[s],\\ Do(\delta_1;\delta_2, s, s') \stackrel{def}{=} \exists s''.Do(\delta_1, s, s'') \wedge Do(\delta_2, s'', s'),\\ Do(\delta_1 \mid \delta_2, s, s') \stackrel{def}{=} Do(\delta_1, s, s') \vee Do(\delta_2, s, s'),\\ Do((\pi x)\delta(x), s, s') \stackrel{def}{=} \exists x.Do(\delta(x), s, s'),\\ Do(\delta^*, s, s') \stackrel{def}{=} \forall P.[\forall s_1 P(s_1, s_1) \wedge\\ \quad \forall s_1, s_2, s_3(Do(\delta, s_1, s_2) \wedge P(s_2, s_3) \supset P(s_1, s_3))]\\ \qquad \supset P(s, s'),\end{array}$$

where A is a primitive action, $Poss(A, s)$ the action precondition predicate, and $\varphi[s]$ the result of restoring s as the situation argument to the fluents in φ.

As we mentioned, the macro $Do(\delta, s, s')$ expands into a formula that is in general second-order. With nested iterations, the formula can get rather complicated. Consider the following complex action $g = (\pi x)(A(x) \mid (\pi x)B(x)^*)^*$. Let $g_1 = (A(x) \mid (\pi x)B(x)^*)^*$, $g_2 = A(x) \mid (\pi x)B(x)^*$, $g_3 = (\pi x)B(x)^*$, and $g_4 = B(x)^*$. Then

$$\begin{array}{l}Do(g, s, s') \stackrel{def}{=} \exists x.Do(g_1, s, s'),\\ Do(g_1, s, s') \stackrel{def}{=} \forall P.\{\forall s_1 P(s_1, s_1) \wedge\\ \quad \forall s_1, s_2, s_3(P(s_1, s_2) \wedge Do(g_2, s_2, s_3) \supset P(s_1, s_3))\}\\ \qquad \supset P(s, s'),\\ Do(g_2, s, s') \stackrel{def}{=} Do(A(x), s, s') \vee Do(g_3, s, s'),\\ Do(A(x), s, s') \stackrel{def}{=} Poss(A(x), s) \wedge s' = do(A(x), s),\\ Do(g_3, s, s') \stackrel{def}{=} \exists x.Do(g_4, s, s'),\\ Do(g_4, s, s') \stackrel{def}{=} \forall P.\{\forall s_1 P(s_1, s_1) \wedge\\ \quad \forall s_1, s_2, s_3(P(s_1, s_2) \wedge Do(B(x), s_2, s_3) \supset P(s_1, s_3))\}\\ \qquad \supset P(s, s'),\\ Do(B(x), s, s') \stackrel{def}{=} Poss(B(x), s) \wedge s' = do(B(x), s).\end{array}$$

As one can see, when fully expanded, $Do(g, s, s')$ is a second-order sentence with nested second-order quantification.

Golog Do macro - a first-order definition

We now show that given that we already have a second-order induction axiom for situations, the macros can actually be defined in first-order terms.

Obviously, the question is how to expand $Do(\delta^*, s, s')$. The second-order sentence in the previous section defines it to be the transitive closure of $Do(\delta, s, s')$. If we introduce a new predicate denoting this transitive closure, then it can be defined inductively just like $<$:

$$Do(\delta^*, s, s') \stackrel{def}{=} P_\delta(\vec{x}, s, s'), \quad (4)$$

$$\begin{array}{l}P_\delta(\vec{x}, s, s') \equiv s = s' \vee\\ \quad \exists s''.s < s'' \leq s' \wedge Do(\delta, s, s'') \wedge P_\delta(\vec{x}, s'', s'),\end{array} \quad (5)$$

where P_δ a new predicate, and $\vec{x}$ the tuple of free variables in δ. Notice that $P_\delta(\vec{x}, s, s')$ is recursive on $P_\delta(\vec{x}, s'', s')$ for s'' such that $s < s'' \leq s'$, making it well-defined according to the "distance" between s and s'.

Our proposal is then to replace the macro definition of $Do(\delta^*, s, s')$ in the previous section by (4) and add the axiom (5) whenever the macro is expanded (see Theorem 1 below).

As an example, consider again the complex action g given above. We have

$$\begin{array}{l}Do(g, s, s') \stackrel{def}{=} \exists x.Do(g_1, s, s'),\\ Do(g_1, s, s') \stackrel{def}{=} P_{g_2}(x, s, s'),\\ P_{g_2}(x, s, s') \equiv s = s' \vee\\ \quad \exists s''.s < s'' \leq s' \wedge Do(g_2, s, s'') \wedge P_{g_2}(x, s'', s'),\\ Do(g_2, s, s') \stackrel{def}{=} Do(A(x), s, s') \vee Do(g_3, s, s'),\\ Do(A(x), s, s') \stackrel{def}{=} Poss(A(x), s) \wedge s' = do(A(x), s),\\ Do(g_3, s, s') \stackrel{def}{=} \exists x.Do(g_4, s, s'),\\ Do(g_4, s, s') \stackrel{def}{=} P_{B(x)}(x, s, s'),\\ P_{B(x)}(x, s, s') \equiv s = s' \vee\\ \quad \exists s''.s < s'' \leq s' \wedge Do(B(x), s, s'') \wedge P_{B(x)}(x, s'', s'),\\ Do(B(x), s, s') \stackrel{def}{=} Poss(B(x), s) \wedge s' = do(B(x), s).\end{array}$$

Thus the axioms about the two extra predicates are

$$\begin{array}{l}P_{g_2}(x, s, s') \equiv s = s' \vee\\ \quad \exists s''\{s < s'' \leq s' \wedge P_{g_2}(x, s'', s') \wedge\\ \quad [Poss(A(x), s) \wedge s'' = do(A(x), s)] \vee \exists x.P_{B(x)}(x, s, s'')\},\\ P_{B(x)}(x, s, s') \equiv s = s' \vee\\ \quad \exists s''.s < s'' \leq s' \wedge P_{B(x)}(x, s'', s') \wedge\\ \quad [Poss(B(x), s) \wedge s'' = do(B(x), s)].\end{array}$$

To illustrate how to reason with these axioms, suppose there are two objects a and b, and that $Poss(A(a), S_0)$ and $Poss(B(b), do(A(a), S_0))$ hold. We show that

$$Do(g, S_0, do(B(b), do(A(a), S_0)))$$

holds. It is easy to see that this follows from

$$\exists x.P_{g_2}(x, S_0, S_2),$$

where $S_2 = do(B(b), S_1)$ and $S_1 = do(A(a), S_0)$. This follows from

$$Poss(A(a), S_0) \wedge P_{g_2}(a, S_1, S_2).$$

The first conjunct is assumed to be true. The second conjunct follows from

$$\exists x.P_{B(x)}(x, S_1, S_2).$$

This is proved by showing $P_{B(x)}(b, S_1, S_2)$, which follows from

$$Do(B(b), S_1, S_2),$$

which is true because $Poss(B(b), S_1)$ is assumed to be true.

Now suppose that there is another primitive action $C(x)$. Furthermore, assume the unique names axioms for actions, $C(x) \neq A(y) \neq B(z)$. Then we can show that $Do(g, S_0, do(C(a), S_0))$ does not hold:

$$\neg Do(g, S_0, S_3),$$

where $S_3 = do(C(a), S_0)$, follows from

$$\forall x.\neg P_{g_2}(x, S_0, S_3),$$

which is equivalent to

$$\forall x \forall s.\neg\{S_0 < s \leq S_3 \wedge P_{g_2}(x, s, S_3) \wedge$$
$$[Poss(A(x), S_0) \wedge s = do(A(x), S_0)] \vee \exists x.P_{B(x)}(x, S_0, s)\}$$

which is equivalent to

$$\forall x \forall s.S_0 < s \leq S_3 \supset$$
$$[\neg[Poss(A(x), S_0) \wedge s = do(A(x), S_0)] \wedge$$
$$\neg\exists x.P_{B(x)}(x, S_0, s)]\}.$$

Because S_3 is the only s that can make $S_0 < s \leq S_3$ true, thus the above assertion is equivalent to

$$\forall x.\neg[Poss(A(x), S_0) \wedge S_3 = do(A(x), S_0)] \wedge$$
$$\neg\exists x.P_{B(x)}(x, S_0, S_3).$$

But $S_3 \neq do(A(x), S_0)$ for any x, thus the above assertion is equivalent to

$$\forall x.\neg P_{B(x)}(x, S_0, S_3),$$

which is equivalent to

$$\forall x.\neg[Poss(B(x), S_0) \wedge S_3 = do(B(x), S_0)],$$

which is equivalent to

$$\forall x.S_3 \neq do(B(x), S_0),$$

which is always true as $S_3 = do(C(a), S_0)$.

The following result is not hard to prove.

Theorem 1 *Let $Do1(\delta, s, s')$ be the macro operator defined in [Levesque et al., 1997] (given in the previous section), and $Do2(\delta, s, s')$ the modification to $Do1$ as described above. For any Golog complex action δ, we have that*

$$\Sigma \cup T_\delta \models \forall \vec{x}, s, s'.Do1(\delta, s, s') \equiv Do2(\delta, s, s'),$$

where $\vec{x}$ is the tuple of free variables in δ, and T_δ is the set of axioms about P_{δ_1} for any δ_1 such that δ_1^* occurs in δ.

Notice that for a program like

$$\delta = [A(a)]^*; [(\pi x)A(x)]^*; (\pi x)A(x)^*,$$

T_δ would contain axioms about $P_{A(a)}(s, s')$, $P_{(\pi x)A(x)}(s, s')$, and $P_{A(x)}(x, s, s')$.

Procedures

Golog programs can have recursively defined procedures in the following form [Levesque et al., 1997]:

proc $P_1(\vec{v}_1)\delta_1$ **endProc** ; ...; **proc** $P_n(\vec{v}_n)\delta_n$ **endProc** ; δ_0.

Here P_i, $1 \leq i \leq n$, are procedures, and δ_0 is the main program. The procedure bodies and the main program δ_i, $0 \leq i \leq n$, are complex actions, extended by procedure calls. In the main program δ_0, a procedure call $P_i(\vec{t}_i)$ is treated like an action:

$$Do(P_i(\vec{t}_i), s, s') \stackrel{def}{=} P_i(\vec{t}_i[s], s, s'),$$

where $P_i(\vec{v}_i, s, s')$ is a new predicate for the procedure P_i, and $\vec{t}_i[s]$, like $\varphi[s]$, is the tuple of terms obtained by restoring s as the situation arguments in the functional fluents occurring in $\vec{t}_i$. As discussed in [Levesque et al., 1997], this amounts to call by value. These new predicates are defined by second-order sentences in [Levesque et al., 1997] to capture least fixed-point semantics of recursive procedures. Here we axiomatize these predicates using first-order axioms, with the help of some additional new predicates: For each $P_i(\vec{v}_i, s, s')$, we introduce a new one with the same name but with one more situation argument $P_i(\vec{v}_i, s, s', s'')$. Intuitively, the extra argument s'' records the number of times that the procedures have been activated. More precisely, $P_i(\vec{v}_i, s, s', s'')$ holds if in the initial situation s, the procedure P_i can return and yields the new situation s' with at most $|s''|$ number of activations for all procedures, where $|s''|$ is the number of actions in the sequence α such that $s'' = do(\alpha, S_0)$. Thus we have the following axioms:

$$P_i(\vec{v}_i, s, s') \equiv \exists s''.P_i(\vec{v}_i, s, s', s''), \quad (6)$$
$$\neg P_i(\vec{v}_i, s, s', S_0), \quad (7)$$
$$P_i(\vec{v}_i, s, s', do(a, s'')) \equiv Do(\delta_i[s''], s, s'), \quad (8)$$
$$Do(P_i(\vec{t}_i, s''), s, s') \stackrel{def}{=} P_i(\vec{t}_i[s], s, s', s''), \quad (9)$$

where $\delta_i[s'']$ is the result of replacing every procedure call of the form $P_j(\vec{t}_j)$ in δ_i by the call $P_j(\vec{t}_j, s'')$.

Consider the following program in the blocks world from [Levesque et al., 1997] ($maketower(n)$ makes a tower of n blocks; $tower(x, m)$ is true if there is a tower of m blocks whose top one is x; $stack(x, n)$ places n blocks on the tower whose top one is x; $unstack(x, n)$ removes n blocks from the tower whose top is x):

proc $maketower(n)$
  $(\pi x, m)[tower(x, m)?;$
  if $m \leq n$ **then** $stack(x, n - m)$
    else $unstack(x, m - n)$
  endIf]
endProc;

proc $stack(x, n)$
  $n = 0? \mid (\pi y)[put(y, x); stack(y, n - 1)]$
endProc;

proc $unstack(x, n)$

$n = 0? \mid (\pi y)[on(x,y)?; movetotable(x);$
$unstack(y, n-1)]$

endProc;

$maketower(7); \neg(\exists x) on(x, A)?$

We have the following axioms for $maketower(n,s,s')$, $stack(x,n,s,s')$, and $unstack(x,n,s,s')$:

$maketower(n,s,s') \equiv \exists s_1.maketower(n,s,s',s_1),$
$stack(x,n,s,s') \equiv \exists s_1.stack(x,n,s,s',s_1),$
$unstack(x,n,s,s') \equiv \exists s_1.unstack(x,n,s,s',s_1),$
$\neg maketower(n,s,s',S_0),$
$maketower(n,s,s',do(a,s_1)) \equiv \exists x,m.[tower(x,m,s) \wedge$
$\quad m \leq n \supset stack(x,n-m,s,s',s_1) \wedge$
$\quad m > n \supset unstack(x,m-n,s,s',s_1)],$
$\neg stack(x,n,s,s',S_0),$
$stack(x,n,s,s',do(a,s_1)) \equiv (n = 0 \wedge s = s') \vee$
$\quad \exists y,s''.s'' = do(put(y,x),s) \wedge stack(y,n-1,s'',s',s_1),$
$\neg unstack(x,n,s,s',S_0),$
$unstack(x,n,s,s',do(a,s_1)) \equiv (n = 0 \wedge s = s') \vee$
$\quad [\exists y,s''.on(x,y,s) \wedge s'' = do(movetotable(x),s) \wedge$
$\quad unstack(y,n-1,s'',s',s_1)].$

To see how our approach handles cycles, consider the following cyclic procedure:

proc P $\{P\}$ **endProc**
P

Clearly, the call P would never terminate. Our axioms for P are:

$P(s,s') \equiv \exists s_1.P(s,s',s_1),$
$\neg P(s,s',S_0),$
$P(s,s',do(a,s_1)) \equiv P(s,s',s_1)$

By induction using the foundational axioms, it is easy to see that these axioms entail $\forall s,s'.s_1.\neg P(s,s',s_1)$, thus $\forall s,s'.\neg P(s,s')$.

Again the following result is not hard to see.

Theorem 2 *Given a Golog program below*

proc $P_1(\vec{v}_1)\delta_1$ **endProc** ; ...; **proc** $P_n(\vec{v}_n)\delta_n$ **endProc** ; δ_0

we have that

$$\Sigma \cup T_{\delta_0} \models \forall \vec{x},s,s'.Do1(\delta_0,s,s') \equiv Do2(\delta_0,s,s'),$$

where $\vec{x}$ is the tuple of free variables in δ, $Do1(\delta_0,s,s')$ and $Do2(\delta_0,s,s')$ the macro operators defined in [Levesque et al., 1997] and this paper, respectively, and T_{δ_0} the set of axioms about the new predicates introduced in defining $Do2$, including those for P_δ for each δ such that δ^* occurs in some δ_i, $0 \leq i \leq n$, and (6) - (9) for each P_i.

Concurrent processes

ConGolog [De Giacomo, Lespérance, and Levesque, 2000] extends Golog by adding some concurrency operators that allows two programs to be executed concurrently. Instead of the Do macro, the main construct is a transition predicate $Trans(\delta,s,\delta',s')$ which says one way to do δ in s is to perform a primitive or a testing action that will lead to situation s' with δ' as the remaining program to be executed. For example, $Trans(a;b,s,b,do(a,s))$ holds because performing the sequence $a;b$ in s will result in performing a, which leads to $do(a,s)$ with b as the remaining program to do.

In [De Giacomo, Lespérance, and Levesque, 2000], the axiomatization of $Trans$ requires programs be reified and quantified over. For instance, the axiom for sequence is

$$\begin{aligned} &Trans(\delta_1;\delta_2,s,\delta',s') \equiv \\ &\quad \exists\gamma.\delta' = (\gamma;\delta_2) \wedge Trans(\delta_1,s,\gamma,s') \vee \\ &\quad Final(\delta_1,s) \wedge Trans(\delta_2,s,\delta',s'), \end{aligned}$$

where $Final(\delta,s)$ is a predicate which says that δ has finished in s.

Using $Trans$, and by quantifying over programs, Do is then defined as

$$Do(\delta,s,s') \stackrel{def}{=} \exists\delta'.Trans^*(\delta,s,\delta',s') \wedge Final(\delta',s'),$$

where $Trans^*$ is the transitive closure of $Trans$.

Here we show that even for concurrent processes, the Do macro can be defined using first-order axioms only, and without having to reify and quantify over programs.

Consider concurrent execution by interleaving $\delta_1 \parallel \delta_2$. Executing this program in s can result in s' if there is a sequence of primitive actions, some from δ_1 and the rest from δ_2, that can be executed in s to yield s'. For example, there are six possible ways to interleave $(P?;A)$ and $(B;A)$, resulting in the following four possible sequences of primitive and test actions:

$$(P?;A;B;A), (P?;B;A;A), (B;A;P?;A), (B;P?;A;A).$$

If we ignore the test actions, as they do not result in changes of situations, we then get two possible sequences of primitive actions: $[A,B,A]$ and $[B,A,A]$. Notice that these are *possible* interleavings. Not all of them may come to fruit. For example, $P?$ may not be true initially, thus cannot be performed as the first action.

If we are told that $[A,B,A]$ is the actual sequence of actions performed, then we know that the first action A must come from the process $(P?;A)$, and the last from $(B;A)$. However, if $[B,A,A]$ is the actual sequence, then for the two A's in the sequence, we do not know from which process each of them come.

To keep track of which action comes from which process in a sequence of primitive actions, we introduce two special primitive actions Y (for yes), and N (for no). Situations consisting of only these two actions are called "schedules':

$sch(S_0),$
$sch(do(a,s)) \equiv sch(s) \wedge (a = Y \vee a = N).$

For example, the schedule $do(Y,do(N,do(Y,do(Y,S_0))))$ $(do([Y,Y,N,Y],S_0))$ tells a process to first execute two actions, then wait for the other process to execute one action,

then execute one more action. Referring back to our earlier example, if we know that $[B, A, A]$ is the actual sequence, then for process $(P?; A)$, there can be only two possible schedules: $do([N, Y, N], S_0)$ or $do([N, N, Y], S_0)$. The first implies that the first A in the sequence $[B; A; A]$ comes from the process, and the second that the second A in the sequence comes from it. Notice that our "schedule situations" only tell when to execute primitive actions while ignore test actions. For process $(P?; A)$ to be successfully executed, we know that P must be tested and succeeded sometime before A is executed, but we do not care exactly when it was actually tested.

These special schedule situations will be used to model possible interleavings. Before we proceed, we first introduce some useful operators about situations:

- We say that two situations are of equal length, written $el(s, s')$, if the number of actions in s is the same as that in s':

$$el(S_0, S_0),$$
$$\neg el(do(a, s), S_0),$$
$$el(do(a, s), do(b, s')) \equiv el(s, s').$$

- We say one schedule s_1 is earlier than s_2, written $earlier(s_1, s_2)$, if at least one action is scheduled earlier in s_1:

$$earlier(s_1, s_2) \equiv \exists s_3.do(Y, s_3) \leq s_1 \wedge do(N, s_3) \leq s_2.$$

- A schedule can be split into two: $split(s, s_1, s_2)$ holds if for any action, it is scheduled to be performed in s iff it is scheduled to be performed either in s_1 or s_2, but not both:

$$split(S_0, S_0, S_0),$$
$$split(do(N, s), s_1, s_2) \equiv \exists s_3, s_4.split(s, s_3, s_4) \wedge$$
$$s_1 = do(N, s_3) \wedge s_2 = do(N, s_4),$$
$$split(do(Y, s), s_1, s_2) \equiv \exists s_3, s_4.split(s, s_3, s_4) \wedge$$
$$[(s_1 = do(Y, s_3) \wedge s_2 = do(N, s_4)) \vee$$
$$(s_1 = do(N, s_3) \wedge s_2 = do(Y, s_4))].$$

  It is easy to see that splitting preserves the length of the schedule:

$$sch(s) \wedge split(s, s_1, s_2) \supset el(s, s_1) \wedge el(s, s_2).$$

- We have two special schedules: $Full(s, s')$ if the schedule from s to s' is full (all "Y"), and $Empty(s, s')$ if the schedule from s to s' is empty (all "N"):

$$Full(s, s') \equiv \forall a, s_1(s < do(a, s_1) \leq s' \supset a = Y),$$
$$Empty(s, s') \equiv \forall a, s_1(s < do(a, s_1) \leq s' \supset a = N).$$

  Notice that $Full(s, s)$ and $Empty(s, s)$ are always true.

ConGolog has three main concurrency operators: $\delta_1 \parallel \delta_2$ (execute δ_1 and δ_2 concurrently by interleaving), $\delta_1 \gg \delta_2$ (interleaving δ_1 and δ_2 with priority to δ_1), and $\delta^{\parallel}$ (interleave δ any number of times).

The prioritized concurrency $\delta_1 \gg \delta_2$ needs special attention. Consider

$$((A; fail?) \mid B) \gg C.$$

Suppose that initially both A and C are executable, but B is not. Suppose further that if C is executed, then B becomes executable. According to the semantics in [De Giacomo, Lespérance, and Levesque, 2000], the above program cannot be executed as $((A; fail?) \mid B)$ has priority, so its first action, which is A, is executed. But once A is executed, it is at a dead end. In this view, if a process has priority, then its first executable action is performed, regardless whether executing that action can lead to a successful execution of this process. However, if we insist on only executing actions that can lead to termination, then the process with lower priority, C, would be executed first as the process with higher priority cannot be executed. After C is performed, B from the other process can then be executed.

Which interpretation is more appropriate depends on the applications. We'll axiomatize both. In our framework, it is easier to axiomatize the latter, so we do it first.

Using schedule situations, we extend the Do macro to concurrency as follows: $Do(\delta, s, s')$ if with a full schedule, executing δ in s can lead to s':

$$Do(\delta, s, s') \stackrel{def}{=} \exists s_1.Full(S_0, s_1) \wedge el(s', s_1) \wedge Do(\delta, s, s', s_1), \tag{10}$$

where $Do(\delta, s, s', s_1)$ is a new macro meaning that executing the sequence of actions from s to s' according to schedule s_1 is a valid way to execute δ in s:

$$Do(A, s, s', s_1) \stackrel{def}{=} \exists s_2, s_3, s''.s < do(A, s'') \leq s' \wedge$$
$$s_2 < do(Y, s_3) \leq s_1 \wedge el(s, s_2) \wedge el(s'', s_3) \wedge$$
$$Empty(s_2, s_3) \wedge Empty(do(Y, s_3), s_1) \wedge Poss(A, s''),$$

$$Do(\varphi?, s, s', s_1) \stackrel{def}{=} \exists s_2, s''.s \leq s'' \leq s' \wedge$$
$$s_2 \leq s_1 \wedge el(s, s_2) \wedge Empty(s_2, s_1) \wedge \varphi[s''],$$

$$Do(\delta_1; \delta_2, s, s', s_1) \stackrel{def}{=} \exists s''.s_2.s_2 \leq s_1 \wedge el(s_2, s'') \wedge$$
$$Do(\delta_1, s, s'', s_2) \wedge Do(\delta_2, s'', s', s_1),$$

$$Do(\delta_1 \mid \delta_2, s, s', s_1) \stackrel{def}{=}$$
$$Do(\delta_1, s, s', s_1) \vee Do(\delta_2, s, s', s_1),$$

$$Do((\pi x)\delta(x), s, s', s_1) \stackrel{def}{=} \exists x.Do(\delta(x), s, s', s_1),$$

$$Do(\delta(\vec{x})^*, s, s', s_1) \stackrel{def}{=} P_\delta(\vec{x}, s, s', s_1),$$
$$P_\delta(\vec{x}, s, s', s_1) \equiv s = s' \vee$$
$$\exists s''.s_2.s < s'' \leq s' \wedge s_2 \leq s_1 \wedge el(s_2, s'') \wedge$$
$$Do(\delta, s, s'', s_2) \wedge P_\delta(\vec{x}, s'', s', s_1),$$

$$Do(\delta_1 \parallel \delta_2, s, s', s_1) \stackrel{def}{=} \exists s_2, s_3.split(s_1, s_2, s_3) \wedge$$
$$Do(\delta_1, s, s', s_2) \wedge Do(\delta_2, s, s', s_3),$$

$$Do(\delta_1 \gg \delta_2, s, s', s_1) \stackrel{def}{=} \exists s_2, s_3\{split(s_1, s_2, s_3) \wedge$$
$$Do(\delta_1, s, s', s_2) \wedge Do(\delta_2, s, s', s_3) \wedge$$
$$\neg\exists s_4, s_5, s_6, s''[split(s_1, s_4, s_5) \wedge earlier(s_4, s_2) \wedge$$
$$s_6 < s_4 \wedge s_6 < s_2 \wedge el(s_6, s) \wedge Do(\delta_1, s, s'', s_4)]\},$$

$$Do(\delta(\vec{x})^{\parallel}, s, s', s_1) \stackrel{def}{=} s = s' \vee P_\delta^{\parallel}(\vec{x}, s, s', s_1),$$
$$\exists s_2(s_2 \leq s_1 \wedge el(s, s_2) \wedge Empty(s_2, s_1)) \supset$$
$$P_\delta^{\parallel}(\vec{x}, s, s', s_1),$$

$$\neg\exists s_2(s_2 \leq s_1 \wedge el(s, s_2) \wedge Empty(s_2, s_1)) \supset$$
$$[P_\delta^{\|}(\vec{x}, s, s', s_1) \equiv$$
$$\exists s_2, s_3.split(s_1, s_2, s_3) \wedge s_1 \neq s_2 \wedge P_\delta^{\|}(\vec{x}, s, s', s_2) \wedge$$
$$Do(\delta(\vec{x}), s, s', s_3)].$$

Notice that for $Do(\delta, s, s', s_1)$ to make sense, the length of the terminating situation s' should be the same as the length of the schedule s_1, and only the segment of the schedule from s to s' is relevant. Thus, for example,

$$Do(\delta, do(a, S_0), do(b, do(a, S_0)), do(Y, do(N, S_0)))$$

holds iff

$$Do(\delta, do(a, S_0), do(b, do(a, S_0)), do(Y, do(Y, S_0)))$$

holds as the two schedules are the same as far as the last action is concerned. We explain the first two definitions in more details below:

- For a primitive action A, $Do(A, s, s', s_1)$ holds if the schedule s_1 has exactly one occurrence of "Y", and at that point, the corresponding action in s' is A. Reformatted in terms of sequences of actions, the definition says that $Do(A, s, s', s_1)$ holds if for some sequences of actions α, γ, β_i, and γ_i, $i = 1, 2$, we have:
$$s = do(\alpha, S_0),$$
$$s' = do(\beta_1, do(A, do(\beta_2, s))),$$
$$s_1 = do(\gamma_1, do(Y, do(\gamma_2, do(\gamma, S_0)))),$$
$$|\alpha| = |\gamma|,$$
$$|\beta_i| = |\gamma_i|, i = 1, 2,$$
$$a \in \gamma_i \supset a = N, i = 1, 2.$$

- For a test action, $Do(\varphi?, s, s', s_1)$ holds if the schedule s_1 is empty, and φ is true somewhere between s and s'. Again, it is easier to understand in terms of sequences of actions. The definition says that $Do(\varphi, s, s', s_1)$ holds if for some sequences of actions α, β, and γ, we have:
$$\exists s''.s \leq s'' \leq s' \wedge \varphi[s''],$$
$$s = do(\alpha, S_0),$$
$$s_1 = do(\gamma, do(\beta, S_0)),$$
$$|\alpha| = |\beta|,$$
$$a \in \gamma \supset a = N.$$

As an example, consider again the following prioritized concurrent program:

$$\delta = ((A; fail?) \mid B) \gg C,$$

where A, B, and C are primitive actions, and $fail$ is a fluent that is never true. Suppose that $Poss(A, S_0)$, $\neg Poss(B, S_0)$, $Poss(C, S_0)$, and $Poss(B, do(C, S_0))$. We show that

$$Do(\delta, S_0, do(B, do(C, S_0))).$$

We need to prove

$$Do(\delta, S_0, do(B, do(C, S_0)), do(Y, do(Y, S_0))).$$

Let $\delta_1 = (A; fail?) \mid B$. The above assertion follows from the following three assertions:

$$Do(\delta_1, S_0, do(B, do(C, S_0)), do(Y, do(N, S_0))), \quad (11)$$
$$Do(C, S_0, do(B, do(C, S_0)), do(N, do(Y, S_0))), \quad (12)$$
$$\neg\exists s_1, s_2, s.split(do(Y, do(Y, S_0)), s_1, s_2) \wedge$$
$$earlier(s_1, do(Y, do(N, S_0))) \wedge Do(\delta_1, S_0, s, s_1). \quad (13)$$

The first assertion (11) follows from

$$Do(B, S_0, do(B, do(C, S_0)), do(Y, do(N, S_0))),$$

which follows from the following facts:

$$el(S_0, S_0),$$
$$el(do(C, S_0), do(N, S_0)),$$
$$Empty(S_0, do(N, S_0)),$$
$$Empty(do(Y, do(N, S_0)), do(Y, do(N, S_0))),$$
$$Poss(B, do(C, S_0)).$$

The second assertion (12) follows easily from the following facts:

$$el(S_0, S_0),$$
$$Empty(S_0, S_0),$$
$$Empty(do(Y, S_0), do(N, do(Y, S_0))),$$
$$Poss(C, S_0).$$

For the last assertion (13), notice first that for

$$split(do(Y, do(Y, S_0)), s_1, s_2) \wedge earlier(s_1, do(Y, do(N, S_0)))$$

to be true, s_1 must be either $do(Y, do(Y, S_0))$ or $do(N, do(Y, S_0))$. But in either case, there is no s such that $Do(\delta_1, S_0, s, s_1)$ can be true.

Notice that under ConGolog's semantics [De Giacomo, Lespérance, and Levesque, 2000], there is no s such that $Do(\delta, S_0, s)$ can be true. This shows that our treatment of the prioritized concurrency operator $\gg$ is different from the one in ConGolog [De Giacomo, Lespérance, and Levesque, 2000]. If we ignore operator $\gg$, then our semantics is equivalent to the one in [De Giacomo, Lespérance, and Levesque, 2000]. Before we prove this, we first prove some formal properties about our axiomatization.

We mentioned that in $Do(\delta, s, s', s_1)$, only the part of the schedule between s and s' matters. We now make this precise.

We say that two situations s_1 and s_2 are the same between s and s', written $eq(s_1, s_2, s, s')$, if there are sequences $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ of actions such that

- $s' = do(\alpha_1, s)$,
- $s_1 = do(\alpha_3, do(\alpha_2, s_3))$ and $s_2 = do(\alpha_4, do(\alpha_2, s_4))$,
- $|\alpha_1| = |\alpha_2|$,
- s_3, s_4, and s are of equal length.

In terms of situation calculus formulas, we have

$$eq(s_1, s_2, s, s') \stackrel{def}{=}$$
$$\exists s_3, s_4, s_5, s_6.s_3 \leq s_1 \wedge s_4 \leq s_2 \wedge el(s_3, s) \wedge el(s_4, s) \wedge$$
$$s_5 \leq s_1 \wedge s_6 \leq s_2 \wedge el(s_5, s') \wedge el(s_6, s') \wedge$$
$$\forall a, s_7, s_8.[s_3 \leq s_7 < s_5 \wedge s_4 \leq s_8 < s_6 \wedge el(s_7, s_8) \supset$$
$$do(a, s_7) \leq s_5 \equiv do(a, s_8) \leq s_6]$$

Lemma 1 *Under our axiomatization, we have that for any δ, s, s', s_1, s_2, if s', s_1, and s_2 are all of equal length, and the schedules s_1 and s_2 are the same between s and s', then $Do(\delta, s, s', s_1)$ iff $Do(\delta, s, s', s_2)$:*

$$s \leq s' \wedge el(s', s_1) \wedge el(s', s_2) \wedge eq(s_1, s_2, s, s') \supset Do(\delta, s, s', s_1) \equiv Do(\delta, s, s', s_2).$$

We mentioned that our formalization of prioritized concurrency $\gg$ is different from that in [De Giacomo, Lespérance, and Levesque, 2000]. We now show that when a program does not have this operator, and does not have the special nil action used in [De Giacomo, Lespérance, and Levesque, 2000], then our axiomatization agrees with that in [De Giacomo, Lespérance, and Levesque, 2000].

Theorem 3 *Let δ be a ConGolog complex action without the prioritized concurrency operator $\gg$, and without the special action nil. Let $Do1$ and $Do2$ be the macro operators defined in [De Giacomo, Lespérance, and Levesque, 2000] and here in this paper, respectively. We have*

$$\Sigma \cup T_1 \cup T_2 \models \forall \vec{x}, s, s'.Do1(\delta, s, s') \equiv Do2(\delta, s, s'),$$

where $\vec{x}$ is the tuple of the free variables in δ, T_1 the set of axioms about $Trans$, $Trans^*$, and $Final$, and T_2 the set of axioms in this paper about sch, el, split, Full, Empty, P_{δ_1} for each δ_1 such that δ_1^* is in δ, and $P^{\|}_{\delta_1}$ for each δ_1 such that $\delta_1^{\|}$ is in δ.

This theorem is proved using the following lemma

Lemma 2 *Let δ be a ConGolog complex action without $\gg$ and nil. Let $s \leq s'$ be two situations, and $s_2 \leq s_1$ two schedules such that $el(s_2, s)$ and $el(s_1, s')$. Then $Do(\delta, s, s', s_1)$ iff for some sequence $s \leq S_1 \leq S_1' \leq S_2 \leq S_2' \leq \cdots \leq S_n \leq S_n' \leq s'$ of situation terms, and a sequence $\delta = \delta_1, ..., \delta_{n+1}$ of ConGolog complex actions, we have that*

- $Trans(\delta_i, S_i, \delta_{i+1}, S_i')$, $1 \leq i \leq n$;
- $Final(\delta_{n+1}, S_n')$;
- *the sequence $S_1 \leq S_1' \leq S_2 \leq S_2' \leq \cdots \leq S_n \leq S_n'$ conforms with the schedule s_1 starting from s_2: for any s_3, $s_2 < do(Y, s_3) \leq s_1$ iff there is an i such that $el(s_3, S_i)$ and $S_i < S_i'$.*

We have already seen an example program that mentions $\gg$ for which our semantics differs from the one in [De Giacomo, Lespérance, and Levesque, 2000]. We now proceed to show that our approach can also handle the $\gg$ operator as defined in [De Giacomo, Lespérance, and Levesque, 2000]. To do this, we introduce a new predicate $PDo(\delta, s, s', s_1)$ (partially do the program), meaning that δ has been carried out from s to s' according schedule s_1. The new definition of $Do(\delta_1 \gg \delta_2, s, s', s_1)$ is as below:

$$\begin{aligned}
&Do(\delta_1 \gg \delta_2, s, s', s_1) \stackrel{def}{=} \exists s_2, s_3\{split(s_1, s_2, s_3) \wedge \\
&\quad Do(\delta_1, s, s', s_2) \wedge Do(\delta_2, s, s', s_3) \wedge \\
&\quad \neg\exists s_4, s_5, s_6, s''[split(s_1, s_4, s_5) \wedge \\
&\quad earlier(s_4, s_2) \wedge do(Y, s_6) \leq s_4 \wedge do(N, s_6) \leq s_2 \wedge \\
&\quad s < s'' \leq s' \wedge el(do(Y, s_6), s'') \wedge \\
&\quad PDo(\delta_1, s, s'', do(Y, s_6))]\}.
\end{aligned}$$

The new predicate PDo is defined as follows. It is the same as Do on primitive actions, test actions, choices, and concurrencies, with Do replaced by PDo:

$$\begin{aligned}
&PDo(A, s, s', s_1) \equiv \exists s_2, s_3, s''.el(s, s_2) \wedge \\
&\quad \{Empty(s_2, s_1) \vee \\
&\quad [s < do(A, s'') \leq s' \wedge s_2 < do(Y, s_3) \leq s_1 \wedge \\
&\quad el(s'', s_3) \wedge Empty(s_2, s_3) \wedge Empty(do(Y, s_3), s_1) \wedge \\
&\quad Poss(A, s'')]\}, \\
&PDo(\varphi?, s, s', s_1) \equiv \exists s_2.s_2 \leq s_1 \wedge el(s, s_2) \wedge \\
&\quad Empty(s_2, s_1), \\
&PDo(\delta_1 \mid \delta_2, s, s', s_1) \equiv \\
&\quad PDo(\delta_1, s, s', s_1) \vee PDo(\delta_2, s, s', s_1), \\
&PDo((\pi x)\delta(x), s, s', s_1) \equiv \exists x.PDo(\delta(x), s, s', s_1), \\
&PDo(\delta_1 \parallel \delta_2, s, s', s_1) \equiv \exists s_2, s_3.split(s_1, s_2, s_3) \wedge \\
&\quad PDo(\delta_1, s, s', s_2) \wedge PDo(\delta_2, s, s', s_3), \\
&PDo(\delta_1 \gg \delta_2, s, s', s_1) \equiv \exists s_2, s_3\{split(s_1, s_2, s_3) \wedge \\
&\quad PDo(\delta_1, s, s', s_2) \wedge PDo(\delta_2, s, s', s_3) \wedge \\
&\quad \neg\exists s_4, s_5, s_6, s''[split(s_1, s_4, s_5) \wedge \\
&\quad earlier(s_4, s_2) \wedge do(Y, s_6) \leq s_4 \wedge do(N, s_6) \leq s_2 \wedge \\
&\quad s < s'' \leq s' \wedge el(do(Y, s_6), s'') \wedge \\
&\quad PDo(\delta_1, s, s'', do(Y, s_6))]\}.
\end{aligned}$$

Partially executing a sequence $\delta_1; \delta_2$ means either partially executing δ_1 or partially executing δ_2 after δ_1 is executed:

$$\begin{aligned}
&PDo(\delta_1; \delta_2, s, s', s_1) \equiv PDo(\delta_1, s, s', s_1) \vee \exists s'', s_2[\\
&\quad s_2 \leq s_1 \wedge Do(\delta_1, s, s'', s_2) \wedge PDo(\delta_2, s'', s', s_1)].
\end{aligned}$$

Similarly for iterations:

$$\begin{aligned}
&PDo(\delta^*, s, s', s_1) \equiv PDo(\delta, s, s', s_1) \vee \\
&\quad \exists s'', s_2.s_2 \leq s_1 \wedge Do(\delta, s, s'', s_2) \wedge PDo(\delta, s'', s', s_1), \\
&PDo(\delta^{\|}, s, s', s_1) \equiv PDo(\delta, s, s', s_1) \vee \\
&\quad \exists s_2, s_3.[split(s_1, s_2, s_3) \wedge s_3 \neq s_1 \wedge \\
&\quad PDo(\delta, s, s', s_2) \wedge PDo(\delta^{\|}, s, s', s_3)].
\end{aligned}$$

Notice that the last axiom will not lead to a cyclic definition as splitting s_1 into s_2 and s_3, with $s_3 \neq s_1$, will eventually lead to s_3 being an empty schedule.

Related work

If we consider generally the problem of formalizing semantics of programs, there is a huge literature about this in computer science. In particular, Golog and its original semantics was strongly influenced by dynamic logic [Harel, 1979]. The unique feature of Golog is that actions are specified axiomatically by successor state axioms, and the Do macro axiomatizes legal execution paths in the situation calculus.

Baier, Fritz and McIlraith [2007] proposed to translate a Golog program to a basic action theory. Later, Fritz, Baier and McIlraith [2008] extended it to ConGolog. Their motivation was to use Golog and ConGolog to encode domain specific control information in planning. Given a basic action theory D, and a goal G, instead of asking if there is an

executable sequence of actions that can achieve G: whether

$$D \models \exists s.Executable(s) \wedge G[s]$$

holds, they proposed to encode the domain specific control information about D as a ConGolog program δ, and ask if there is a sequence of actions that is a legal execution of δ and can achieve G: whether

$$D \models \exists s.Do(\delta, S_0, s) \wedge G[s].$$

To implement this, they translated D and δ to a new basic action theory D' in a new language, and showed that there is a mapping τ from sequences of actions (situations) in the language of D' to sequences of actions (situations) in the language of D such that $D' \models Executable(S) \wedge G[S]$ iff $D \models Do(\delta, S_0, \tau(S)) \wedge G[\tau(S)]$. Thus, the semantics of δ is captured by D'. While the axioms in D' are also first-order, they use natural numbers. In comparison, our axiomatization uses the same language of the original situation calculus. An interesting question is whether our first-order reformulation of Golog and ConGolog semantics could make the translation from D and δ to D' simpler.

While we consider Golog and ConGolog programs, Gabaldon [2003; 2004] considered "programs" written in temporal logic formulas, and their translation into first-order situation calculus.

Conclusions

We have shown that by using the foundational axioms in the situation calculus, it is possible to give a first-order semantics to Golog and ConGolog programs. In the case of Golog, this is achieved by introducing a new predicate when an iteration is encountered. In the case of ConGolog, this is done by using so-called scheduling situations. The overall message is that once we have second-order induction on the situations, other inductive structures and fixed-points can be defined in first-order terms. We hope the simpler first-order semantics will lead to more effective ways of doing program verification and synthesis in Golog and ConGolog.

Acknowledgments

My thanks to Hector Levesque for his comments on an earlier version of this paper, and for a pointer to a related work. I also thank the reviewers for their helpful comments. This work was supported in part by HK RGC under GRF 616013, and NSFC of China under grant 6137061

Appendix: Proofs

Proof of Theorem 1

Let δ be a complex action, M a model of $\Sigma \cup T_\delta$, and σ a variable assignment for $\vec{x}$, s, and s'. We need to show that

$$M, \sigma \models Do1(\delta, s, s') \equiv Do2(\delta, s, s').$$

This is done by induction on the structure of δ. Notice that $Do1$ and $Do2$ are identical except on iterations, so we need only to prove the inductive step for this construct. We know that

$$M, \sigma \models Do1(\delta^*, s, s')$$

iff there is a sequence of $S_0, .., S_n$, $0 \leq n$, of situation terms, such that under σ, $s = S_0$, $S_n = s'$, and for any $0 \leq i < n$,

$$M, \sigma \models Do1(\delta, S_i, S_{i+1}).$$

On the other hand,

$$M, \sigma \models Do2(\delta^*, s, s')$$

iff

$$M, \sigma \models P_\delta(\vec{x}, s, s').$$

Since M is a model of T_δ, by induction on s', it can be seen that the above holds iff there is a sequence of $S_0, .., S_n$, $0 \leq n$, of situation terms, such that $s = S_0$, $S_n = s'$, and for any $0 \leq i < n$,

$$M, \sigma \models Do2(\delta, S_i, S_{i+1}).$$

Thus the desired result holds by the inductive assumption that

$$M, \sigma \models Do2(\delta, S, S') \equiv Do1(\delta, S, S')$$

for any situation terms S and S'.

Proof of Theorem 2 (Sketch)

Inductively for any complex action δ extended with P_i, $1 \leq i \leq n$, define $Do1^k$ as follows:

$$\begin{aligned}
&P_i^0(\vec{v}_i, s, s') \equiv false,\\
&P_i^k(\vec{v}_i, s, s') \equiv Do1^{k-1}(\delta_i, s, s'),\\
&Do1^k(\delta, s, s') \stackrel{def}{=} Do1(\delta(\vec{P}/\vec{P^k}), s, s'),
\end{aligned}$$

where $\delta(\vec{P}/\vec{P^k})$ is the result of replacing each occurrence of $P_i(\vec{t})$ in δ by $P_i^k(\vec{t})$, for every $1 \leq i \leq n$. The second order definition of $Do1(\delta_0, s, s')$ in [Levesque et al., 1997] captures the least fixed points for P_i, meaning that $Do1(\delta_0, s, s')$ iff for some k, $Do1^k(\delta_0, s, s')$. From this, the result of the theorem follows.

Proof of Lemma 1

Suppose $el(s', s_1)$, $el(s', s_2)$, and $eq(s_1, s_2, s, s')$ hold. This means that s_1 and s_2 are the same after s, not just between s and s', as s' is really the end of s_1 and s_2. Suppose $Do(\delta, s, s', s_1)$. We prove by induction on δ that $Do(\delta, s, s', s_2)$ holds as well. The base cases for primitive actions and test actions are straightforward by noting that s_1 and s_2 are the same after s. Inductively, we have the following cases:

- $\delta = \delta_1; \delta_2$. From $Do(\delta, s, s', s_1)$, we have that for some s'' and s_3, $s_3 \leq s_1$, $el(s_2, s'')$, and

  $$Do(\delta_1, s, s'', s_3) \wedge Do(\delta_2, s'', s', s_1).$$

  hold. Now let $s_4 \leq s_2$ and $el(s_4, s_3)$. By inductive assumptions, we have

  $$Do(\delta_1, s, s'', s_4) \wedge Do(\delta_2, s'', s', s_2),$$

  which implies $Do(\delta, s, s', s_2)$.
- The cases for $\delta_1 \mid \delta_2$ and $(\pi x)\delta_1(x)$ are trivial.

- $\delta = \delta_1(\vec{x})^*$. From $Do(\delta, s, s', s_1)$, we have $P_{\delta_1}(\vec{x}, s, s', s_1)$. We show by induction on s that $P_{\delta_1}(\vec{x}, s, s', s_2)$ holds as well. The base case $s = s'$ is trivial. Inductively, assume $s < s'$ and the result holds for all s'' such that $s < s'' \leq s'$. From $P_{\delta_1}(\vec{x}, s, s', s_1)$, we have that for some s'' and s_3:

$$s < s'' \leq s' \wedge s_3 \leq s_1 \wedge el(s_3, s'') \wedge Do(\delta_1, s, s'', s_3) \wedge P_{\delta_1}(\vec{x}, s'', s', s_1).$$

By inductive assumption, we have

$$s < s'' \leq s' \wedge s_3 \leq s_1 \wedge el(s_3, s'') \wedge Do(\delta_1, s, s'', s_3) \wedge P_{\delta_1}(\vec{x}, s'', s', s_2).$$

Now let $s_4 \leq s_2$ be such that $el(s_4, s_3)$, by inductive assumption on δ_1, we have

$$s < s'' \leq s' \wedge s_4 \leq s_2 \wedge el(s_4, s'') \wedge Do(\delta_1, s, s'', s_4) \wedge P_{\delta_1}(\vec{x}, s'', s', s_2).$$

Thus $P_{\delta_1}(\vec{x}, s, s', s_2)$.

- $\delta = \delta_1 \parallel \delta_2$. From $Do(\delta, s, s', s_1)$, we have for some s_3, s_4 such that $split(s_1, s_3, s_4)$,

$$Do(\delta_1, s, s', s_3) \wedge Do(\delta_2, s, s', s_4).$$

Since s_1 and s_2 are the same after s, we can split s_2 the same way after s: $split(s_2, s_5, s_6)$ so that s_3 and s_5, and s_4 and s_6, respectively, are the same after s. By inductive assumption, we then have

$$Do(\delta_1, s, s', s_5) \wedge Do(\delta_2, s, s', s_6).$$

Thus $Do(\delta, s, s', s_2)$.

- The case for $\delta_1 \gg \delta_2$ is similar to the previous case.
- $\delta = \delta_1(\vec{x})^{\parallel}$. The case for $s = s'$ is trivial. Suppose $s < s'$. From $Do(\delta_1(\vec{x})^{\parallel}, s, s', s_1)$, we have $P^{\parallel}_{\delta_1}(\vec{x}, s, s', s_1)$. We show by induction on the number of "Y" actions in s_1 that $P^{\parallel}_{\delta_1}(\vec{x}, s, s', s_2)$ holds as well. If s_1 is all empty, then s_2 is all empty starting from s:

$$\exists s_3(s_3 \leq s_2 \wedge el(s, s_3) \wedge Empty(s_3, s_2)). \quad (14)$$

By our axiom about $P^{\parallel}$, $P^{\parallel}_{\delta_1}(\vec{x}, s, s', s_2)$ holds. Inductively, there are two cases. Case 1: s_1 is all empty starting from s. Then s_2 is all empty starting from s as well. Case 2: s_1 is not all empty starting from s. Then for some s_3 and s_4 such that $split(s_1, s_3, s_4) \wedge s_1 \neq s_3$, we have

$$P^{\parallel}_{\delta_1}(\vec{x}, s, s', s_3) \wedge Do(\delta_1(\vec{x}), s, s', s_4).$$

We can split s_2 in the same way: $split(s_2, s_5, s_6)$ so that $s_2 \neq s_5$, and s_5 and s_3, and s_6 and s_4, respectively, are the same after s. Now the number of "Y" actions in s_3 must be less than that in s_1. Thus by inductive assumptions,

$$P^{\parallel}_{\delta_1}(\vec{x}, s, s', s_5) \wedge Do(\delta_1(\vec{x}), s, s', s_6).$$

Thus $P^{\parallel}_{\delta_1}(\vec{x}, s, s', s_2)$.

This completes the inductive step, thus the proof of the lemma.

Proof of Lemma 2 (Sketch)

Let $s \leq s'$ be two situations, and $s_2 \leq s_1$ two schedules such that $el(s_2, s)$ and $el(s_1, s')$. We prove by induction on δ. There are two base cases, primitive actions and test actions. If $\delta = A$ is a primitive action, then $Do(\delta, s, s', s_1)$ iff for some s_3 and s'',

$$s < do(A, s'') \leq s' \wedge s_2 < do(Y, s_3) \leq s_1 \wedge el(s'', s_3) \wedge Empty(s_2, s_3) \wedge Empty(do(Y, s_3), s_1) \wedge Poss(A, s'')$$

iff for some s_3 and s'', the following are true

- $s \leq s'' < do(A, s'') \leq s'$,
- $Trans(A, s'', nil, do(A, s'')) \wedge Final(nil, do(A, s''))$,
- the sequence $s'' < do(A, s'')$ conforms with the schedule s_1 starting from s_2: it has exactly one "Y" which occurs at the position marked by s''.

Noting that $Trans(A, S, \delta', S')$ holds iff $\delta' = nil$, $S' = do(A, S)$, and $Poss(A, S)$, this proves the lemma for the case.

The case for $\delta = \varphi?$ is almost identical, noting that $Trans(\varphi?, S, \delta', S')$ iff $\varphi[S]$, $S' = S$, and $\delta' = nil$.

Inductively, there are several cases. Suppose $\delta = \delta^1; \delta^2$. Then $Do(\delta^1; \delta^2, s, s', s_1)$ iff for some s_3, s''

$$s_3 \leq s_1 \wedge el(s_3, s'') \wedge Do(\delta^1, s, s'', s_3) \wedge Do(\delta^2, s'', s', s_1).$$

The result then follows from the inductive assumptions on $Do(\delta^1, s, s'', s_3)$ and $Do(\delta^2, s'', s', s_1)$, and the definition of $Trans$ from [De Giacomo, Lespérance, and Levesque, 2000]. If $\delta = \delta^1 \parallel \delta^2$, then $Do(\delta^1 \parallel \delta^2, s, s', s_1)$ iff for some s_3 and s_4,

$$split(s_1, s_3, s_4) \wedge Do(\delta^1, s, s', s_3) \wedge Do(\delta^2, s, s', s_4).$$

Notice that $split(s_1, s_3, s_4)$ implies that $el(s_1, s_3)$ and $el(s_1, s_4)$. Thus the inductive assumption can apply to $Do(\delta^1, s, s', s_3)$ and $Do(\delta^2, s, s', s_4)$, from which the result follows. The other cases are similar.

Proof of Theorem 3

$Do2(\delta, s, s')$ iff $Do2(\delta, s, s', s_1)$, where s_1 is a full schedule such that $el(s', s_1)$, iff (by Lemma 2) for some sequence $s \leq S_1 \leq S'_1 \leq S_2 \leq S'_2 \leq \cdots \leq S_n \leq S'_n \leq s'$ of situation terms, and a sequence $\delta = \delta_1, ..., \delta_{n+1}$ of ConGolog complex actions, we have that

- $Trans(\delta_i, S_i, \delta_{i+1}, S'_i)$, $1 \leq i \leq n$;
- $Final(\delta_{n+1}, S'_n)$;
- the sequence $S_1 \leq S'_1 \leq S_2 \leq S'_2 \leq \cdots \leq S_n \leq S'_n$ conforms with the the full schedule s_1 starting from s_2, where $s_2 \leq s_1$ and $el(s, s_2)$: for any s_3 such that $s_2 \leq s_3 < s_1$, there is an i such that $el(s_3, S_i)$ and $S_i < S'_i$.

Given that s and s_2 have the same length, and s' and s_1 have the same length, the last condition implies that $S'_i = S_{i+1}$ for every $1 \leq i < n$, $s = S_1$, and $S'_n = s'$. Thus the above condition holds iff $Do1(\delta, s, s')$.

References

Baier, J. A.; Fritz, C.; and McIlraith, S. A. 2007. Exploiting procedural domain control knowledge in state-of-the-art planners. In *ICAPS-2007*, 26–33.

Burgard, W.; Cremers, A. B.; Fox, D.; Hähnel, D.; Lakemeyer, G.; Schulz, D.; Steiner, W.; and Thrun, S. 1999. Experiences with an interactive museum tour-guide robot. *Artificial Intelligence* 114(1-2):3–55.

De Giacomo, G.; Lespérance, Y.; and Levesque, H. J. 2000. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence* 121 (1–2):109–169.

Fritz, C.; Baier, J. A.; and McIlraith, S. A. 2008. ConGolog, sin trans: Compiling congolog into basic action theories for planning and beyond. In *KR-2008*, 600–610.

Funge, J. 2000. Cognitive modeling for games and animation. *Communications of ACM* 43(7):40–48.

Gabaldon, A. 2003. Compiling control knowledge into preconditions for planning in the situation calculus. In *IJCAI-2003*, 1061–1066.

Gabaldon, A. 2004. Precondition control and the progression algorithm. In *KR-2004*, 634–643.

Harel, D. 1979. *First-Order Dynamic Logic*. New York: Springer-Verlag: Lecture Notes in Computer Science 68.

Lespérance, Y.; Levesque, H. J.; and Ruman, S. J. 1997. An experiment in using Golog to build a personal banking assistant. In *Intelligent Agent Systems: Theoretical and Practical Issues, volume 1209 of LNAI*, 27–43. Springer-Verlag.

Levesque, H.; Reiter, R.; Lespérance, Y.; Lin, F.; and Scherl, R. 1997. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming, Special issue on Reasoning about Action and Change* 31:59–84.

Lin, F., and Reiter, R. 1994. State constraints revisited. *Journal of Logic and Computation, Special Issue on Actions and Processes* 4(5):655–678.

Lin, F. 2007. Situation calculus. In van Harmelen, F.; Lifschitz, V.; and Porter, B., eds., *Handbook of Knowledge Representation*. Elsevier.

McCarthy, J. 1968. Situations, actions and causal laws. In Minsky, M., ed., *Semantic Information Processing*. MIT Press, Cambridge, Mass. 410–417.

McIlraith, S. A., and Son, T. C. 2002. Adapting Golog for composition of semantic web services. In *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002)*, 482–496.

Reiter, R. 1993. Proving properties of states in the situation calculus. *Artificial Intelligence* 64:337–351.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press.